

TMSchema: Introduction

- TMSchema: proposal for TMCL Lite
- Schema building blocks
- User perspective
- Behind the scene
- Partial Type Descriptions (PTD)
- PTDs merging and conflicts
- Explicit exceptions
- TMSchema and inference in Topic Maps
- Schema introspection using TMQL
- Differences between TMSchema and OWL
- Summary

TMSchema – proposal for TMCL Lite

- Allows to define basic set of constraints
- Has good introspection
- Supports simple validation
- Can be used easily for building “smart” interfaces
- Can be characterized as “extended OSL”
- Is expressed in general terms (set of nodes with properties)
- Can be represented as topic map constructs
- Can be supported by several syntaxes

Schema building blocks

TopicSchema

BaseNameSchema(type,scope) *

InternalOccurrenceSchema(type,scope) *

ExternalOccurrenceSchema(type,scope) *

PlayRoleSchema(roleType,associationType,scope)*

RoleSchema(roleType)*

OneOfSchema?

RelatedTopicSchema(type,scope)*

AssociationSchema(scope)

RoleSchema(roleType)+

Base Name Schema

BaseNameSchema(type,scope):

cardMin	#Integer
cardMax	#Integer
dataType	#xsd and custom xml schemas
oneOf	# String*
match	# Regular Expression*

Internal Occurrence Schema

InternalOccurrenceSchema(type,scope):

cardMin	# Integer
cardMax	# Integer
dataType	# xsd and custom schemas
one of	# String*
match	# Regular Expression *

External Occurrences Schema

ExternalOccurrenceSchema(type,scope):

cardMin	# Integer
cardMax	# Integer
one of	# URI*
match	# Regular Expression *

Play Role Schema

\$X can play <role> in <association> with <scope>

PlayRoleSchema(role,association,scope):

cardMin	# Integer
cardMax	# Integer
otherPlayers	# RoleSchema*

Role Schema

RoleSchema(role):

cardMin	# Integer
cardMax	# Integer
allPlayersFrom	# Topic* list of Types
somePlayersFrom	# Topic* list of Types
oneOf	# Topic* list of topics

One of Schema

OneOfSchema:

one of

Topic+ - List of Topics

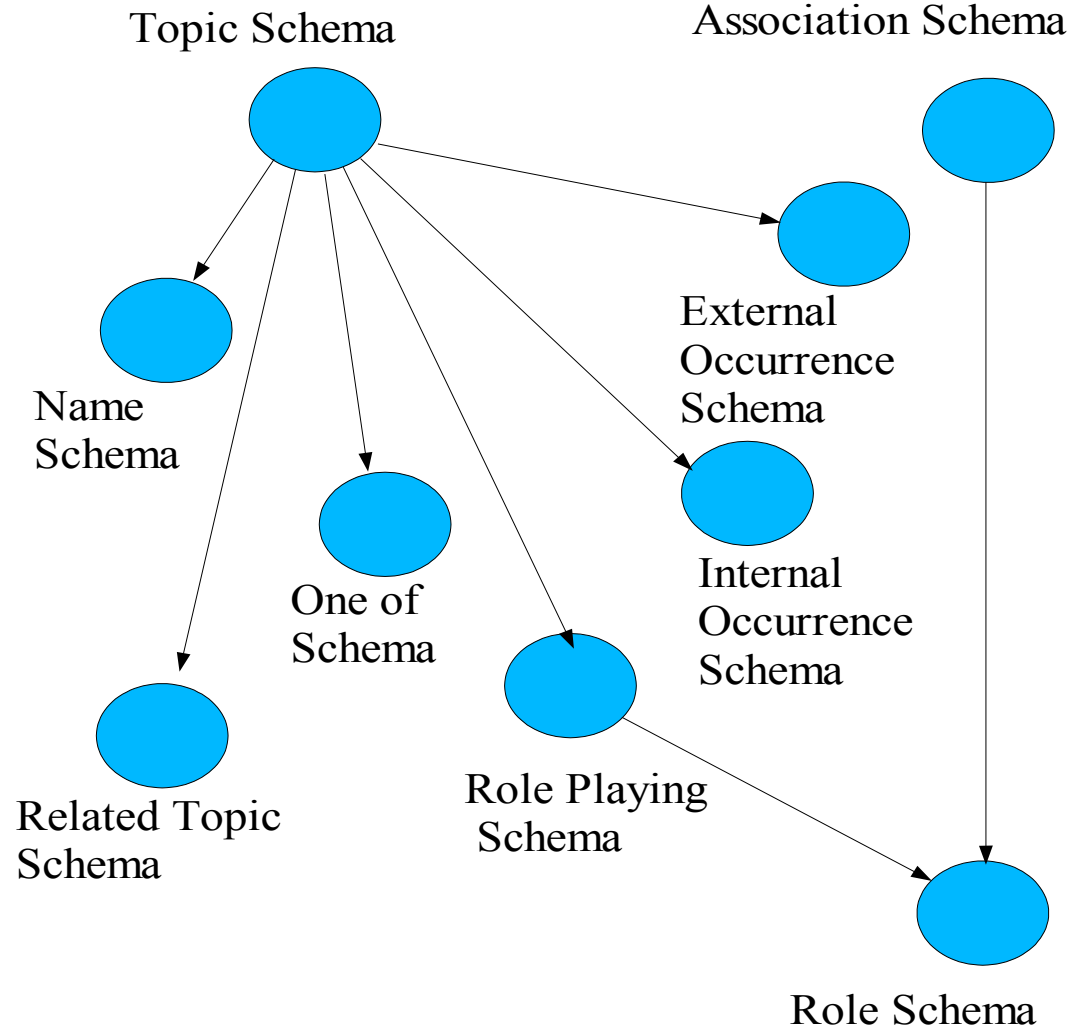
RelatedTopicSchema

If topic represents resource this schema defines possible related topics

RelatedTopicSchema(type,scope)

cardMin	# Integer	
cardMax	# Integer	
allRelatedTopicsFrom	# Topic*	list of Types
someRelatedTopicsFrom	# Topic*	list of Types
oneOf	# Topic*	

Schema Diagram



User Perspective

- Users attach schemas to type or association
- They use one of the TMSchema compatible syntaxes
- Schema is “nice to have” information
- Users define as much as they know/ want to represent in schema
- More precise schema – more support from TM engine
- TM engine can work without schemas

Example

[composer : musician = "Composer"]

baseNameSchema(composer, baseName, {cardMin=1,cardMax=1})

baseNameSchema(composer,baseName/normal, {cardMin=1,cardMax=1})

internalOccurrenceSchema(composer,born,{cardMin=1,cardMax=1,dataType="date"})

externalOccurrnceSchema(composer,website,{cardMin=1})

playRoleSchema(composer,who,bornIn,

{cardMin=1,cardMax=1,

roleSchema(where, {cardMin=1,cardMax=1, allPlayersFrom=[location])

})

Example: Compact form

Same set of constraints can be represented in more compact form

```
[composer : musician = "Composer",
  {
    baseNameSchema(baseName, {cardMin=1,cardMax=1}),
    baseNameSchema(baseName/normal, {cardMin=1,cardMax=1})
    internalOccurrenceSchema(born,{cardMin=1,cardMax=1,dataType="date"})
    externalOccurrnceSchema(composer,website,{cardMin=1})
    playRoleSchema(who,bornIn,
      {
        cardMin=1,cardMax=1,
        roleSchema(where, {cardMin=1,cardMax=1, allPlayersFrom=[location])
      })
  }
]
```

Behind the Scene

- Key Concept – Partial Type Description (PTD)
- Partial Type Description describes available type constraints
- Each type/association can have multiple partial descriptions
- Partial descriptions can come from various sources
- TMSchema interpretation defines mechanism of PTD merging
- Merged PTDs represent “final” type/association schema
- PTDs can be inconsistent
- TMSchema has explicit concept of conflicts
- TMSchema supports conflict resolution process

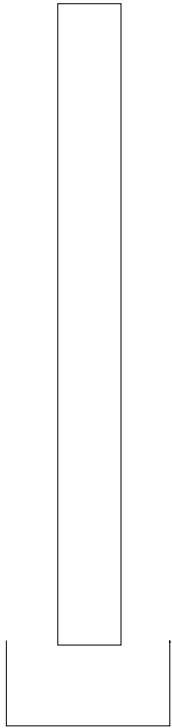
Partial Type Description (PTD)

- PTD represents necessary rule
 - if \$X isInstanceOf \$Y then <some constraints>
- PTD has structure and is represented as Topic or Association schemas
- PTD can be user defined:
 - this is what users define during Topic Map authoring
 - Topic Map author is one of the possible PTD sources
- PTD can be inherited from super type
- PTD can be “inherited” from merged Topic Maps
- PTD can be deduced from other sources

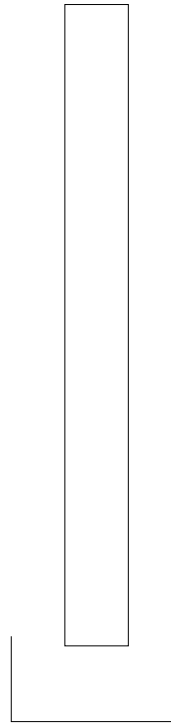
Merging of Partial Type Descriptions

Add constraints
and exceptions

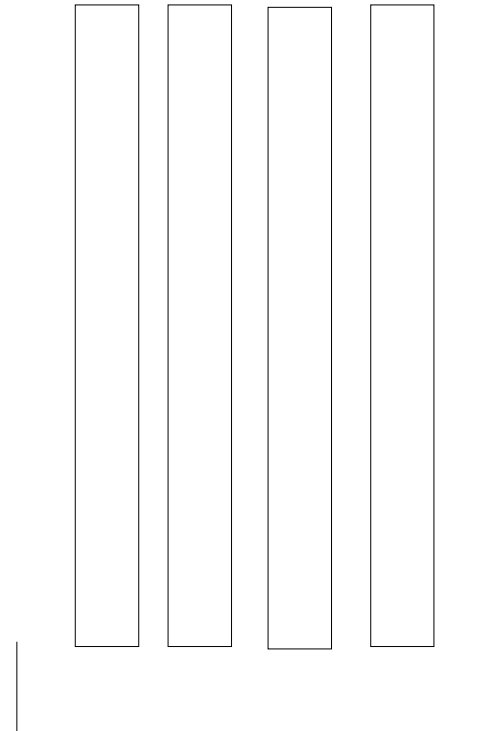
Merging Schemas



Final Schema

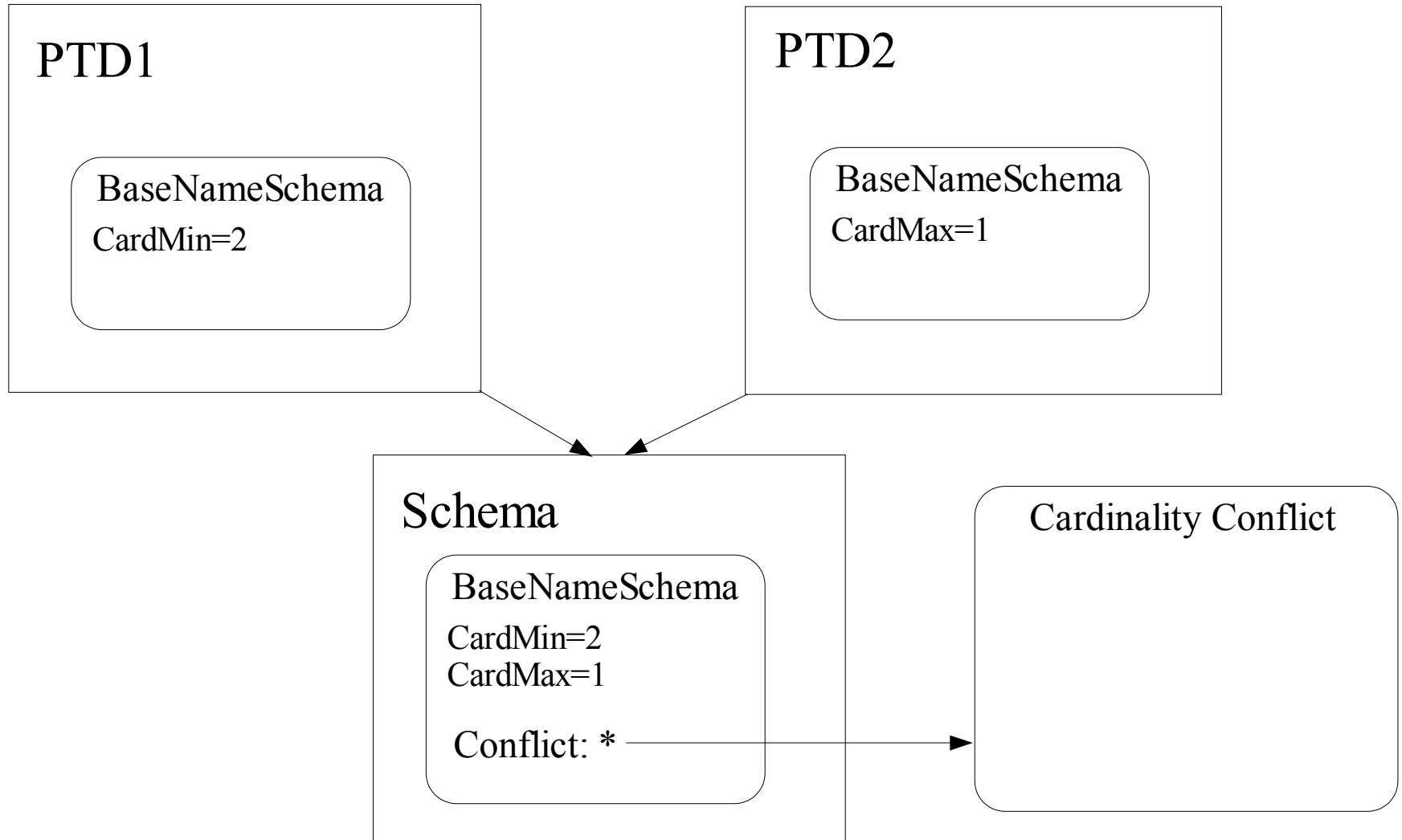


User Defined PTD



Deduced PTDs

Conflicts during PTD merging: Example



Conflicts during PTD merging

- TMSchema engine identifies basic schema conflicts
- Users are responsible for resolving conflicts:
 - ♦ Can modify other PTDs (super types, for example)
 - ♦ Can specify explicit exception
- TMSchema engine can help users with conflict resolution:
 - ♦ By providing constraint justifications
 - ♦ By providing typical conflict resolution advices

Explicit Exceptions

- Users can specify explicit schema exception to prevent a conflict
- It is important when user does not have control over merged topic maps

Example 1:

```
schemaException(composer, musician/baseNameSchema(normal)/cardMin )
```

Prevents propagation of cardMin constraint

Example 2:

```
schemaException(composer, opera_tm/musician/baseNameSchema(normal)/cardMin )
```

Prevents propagation of cardMin constraint from merged opera_tm topic map

TMSchema and inference in Topic Maps

- TMSchema introduces concept of inference in Topic Maps
- Some nodes and properties in Topic Map are created by TMSchema engine
- How can we represent results of inference?
- Proposal: use scope mechanism for representing results of inference

Example:

user defined constraint

```
baseNameSchema(composer, baseName, {cardMin=1,cardMax=1})
```

“proposal” generated by TMSchema engine

```
baseNameSchema(composer, baseName, {cardMin=0,cardMax=1}) / deduced, proposal
```

“final result” after constraint merges generated by TMSchema engine

```
baseNameSchema(composer, baseName, {cardMin=1,cardMax=1}) / deduced, result
```

Schema inference modules

- TMSchema supports following schema inference modules:
 - ◆ SuperType-SubType schema inheritance
 - ◆ Association to Topic schema propagation
 - ◆ Resource to Topic schema propagation

Topic map merging and schemas

- Schemas are copied from source to target topic map during merging
- Assumption: TMSchema engine “knows” about source of each schema

Example:

If TM1 is merged into TM0
and both topic maps have schema for type composer

TMSchema engine can distinguish

TM0/composer/baseNameSchema/cardMin

and

TM1/composer/baseNameSchema/cardMin

Justifications

- Justifications are based on TMSchema inference modules
 - ♦ SuperType-SubType schema inheritance justification
 - ♦ Association to Topic schema propagation justification
 - ♦ Resource to Topic schema propagation justification

Justifications: Example

Constraint: composer/baseNameSchema/cardMin

Value: 1

Aggregation Type: max

Proposals:

Value	Inference	Exceptions
1	SuperType Inheritance (Musician)	OK
1	UserSpecified (TM1/Composer)	OK
0	SuperType Inheritance (Person)	OK

Schema Introspection

- TMQL can be used to query schema
- It is possible to query:
 - ♦ user defined schemas,
 - ♦ deduced schema proposals
 - ♦ and final schema-result

Examples:

```
canPlayRole(composer,who,bornIn,$RoleSchema),  
cardMin($RoleSchema,$Min),  
cardMax($RoleSchema,$Max)
```

```
canPlayRole(composer,$Role,$A,_) / [deduced,result]
```

Differences between TMSchema and OWL

TMSchema:

- is part of TM itself, “external files” are not required
- supports only necessary conditions
- does not do any “auto-classifications”
- relies on “justification based” reasoning
- supports “exceptions”
- does not deal with identity constraints

Reusing OWL Ontologies

- TMSchema will provide TMSchema-based views on existing OWL Ontologies
- It will be possible to “include” and extend OWL Ontologies.
- Views will be based on mapping specifications
- TMSchema engine extracts constraints which are representable in TMSchema

Summary

- TMSchema is a proposal for TMCL Lite
- Supports basic validation and introspection
- Can be extended with Schematron-like rules
- Will include OWL mapping and views
- Can be extended with OWL-like “type definitions”